Interim Report
ONR Grant N00014-93-1-0855

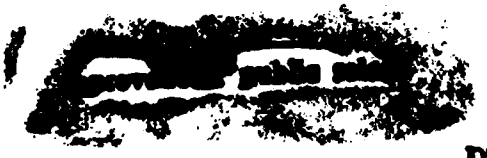
Neural Networks Modeling for Yield Enhancement in Semiconductor Manufacturing

Grant Period:
July 1, 1993 - February 28, 1995

Dr. Jacek M. Zurada, PI
University of Louisville
Louisville, Kentucky 40292

January 31, 1993

This Interim Report covers the period
July 1, 1993 - December 31, 1993



Interim Report
ONR Grant N00014-93-1-0855

Neural Networks Modeling for Yield Enhancement in Semiconductor Manufacturing

This Interim Report covers the period
July 1, 1993 - December 31, 1993

Introduction

Major tasks planned during the first period of the research project have been the analysis of data and evaluation of its statistical properties. Selection of feasible neural network architectures and their feasibility study have been pursued. Also, development of data manipulation software has been facilitated during the initial phase of the project work for the purpose of properly interfacing data with the neural network models.

1. Data Release

The GaAs fabrication data measured under the Microwave/Millimeter Wave Integrated Circuit (MIMIC) Phase 1, Task 4.E Program have been made available to the project team in the initial phase of the project. This has been possible as a result of the data release (Case Number ASC931796).

Available measurement data have been first inspected visually. It has been determined that some values are far exceeding range of measuring devices. Such data were not assumed as credible measurement results and have been considered missing (not taken). This has been found necessary to distinguish missing data from data which are still significant but are distant from their mean values.

2. Data Manipulation and Evaluation Software

Test structures available have been measured across the entire wafer in one test sweep. Computer software has been first written which manipulate the raw measurement data. The reticle ID identified as XXYY, the structure ID within the reticle identified as xxyy, and the measured value are placed in a file. The characters in the filename have been chosen in such a manner that the fabricator, process lot, boule source, wafer number, the characteristics measured, and the processing stage at which the measurement was performed are easily identifiable.

The majority of the characteristics have been measured on the 1x200μm MESFET. The test structure/device (referred to as a *device* from this point on) is at the center of this modeling effort. This device allows for a comprehensive DC and RF characterization of the FET equivalent circuit parameters. For network training purposes we have found it initially desirable

to maintain the density of six training vectors per reticle. Therefore, the data have been taken at 6 distinct locations within the reticle. Training vectors have been formed by assigning each of the non-MESFET characteristics to the nearest-neighbor MESFET. This has resulted in six complete training vectors per reticle.

Training vectors need to be subsequently formed for each of the neural network models. This requires files of training vectors be created as training files, one representing each processing stage, and then for later stages a file is needed containing all measurements for prior fabrication stages.

The manipulation routine initially developed searches through the filenames of characteristics measured on a user-selected wafer. The stage at which the measurement was made is determined by a specific letter in the filename. A large table representing all measurements on the wafer has been formed. Then, for user requested reticles and process stages, files of training/test data vectors are created. Another, more flexible routine is currently in preparation.

A routine is currently under development which calculates the statistics of the input and output parameters. This routine will provide insight as to what form the parameters take, and will identify outliers and statistical moments. The computed statistics will allow to better determine the training condition in the situation of statistical distribution of training and test data.

3. Neural Networks Modeling Software

Inspection of data indicates that feasible architectures for modeling process stages of interest are those of multilayer feedforward perceptrons (MFFNN). In order to fully control the training process and have clearly defined measures of training errors, the MFFNN training software developed in house has been adapted to project needs. The software allows for increased input/output capabilities. Each network component, and both training and test functions of MFFNN can now be controlled and adequately monitored by the modeler, which functions have been rather hard to implement in commercial training packages. The software has been written in C language and can be executed on the Unix or PC platform.

In addition to the adapted training and recall software for MFFNN, sensitivity analysis package has been developed. The software's objective is to delete inputs which are irrelevant for the current modeling effort. The detailed properties of the sensitivity analysis algorithm and its implementation, as well as experimental results are described in the paper appended to this report. The sensitivity software has been verified on a number of cases which show its relevancy for our modeling effort.

4. Preliminary Process Modeling Effort

To determine each of the programs' functionality and the feasibility of the process stage models, preliminary process stage models of two fabrication stages were created. The post-

contact to post-recess first and then post-contact, post-recess, to post-gate process stages were modeled. Initially, data from only one wafer was chosen for modeling. A three by three block of reticles in each of three of the four quadrants of the wafer were selected for modeling. This resulted in 54 (6x9) training/test vectors, representing nine reticles per quadrant. The training vectors were shuffled in a random manner before training and test files were created. The first 40 vectors of the shuffled set were used for network training and the last 14 left for testing the network. Each group of nine reticles was modeled separately. Section 4.1 and 4.2 discuss the results of single quadrant modeling.

4.1 Post-contact to Post-Recess (c=>r or C) Models

The network for this initial model has six inputs, five outputs, and 12 hidden neurons plus the bias input. The size of the hidden layer was determined experimentally by varying the number of hidden neurons and selecting their number which resulted in the lowest training error over a number of training sessions. Graphs of the actual measured and modeled values are presented in Fig. 1. Each of the five diagrams displays a training set model of the stage and it indicates an excellent agreement between the actually measured and MFFNN-modeled data.

Fig. 2 contains graphs representing the training results for each of the outputs when the network is presented the test set. The network now representing the stage model has never seen the test set data before. The error for the network-processed new data is several percent on the average. It can be seen that the developed network models for the C stage provides good prediction of process outputs on the test set. Note that the error appears larger on the figures than it really is since the axes for parameter values do not start at zero values.

The preliminary models for the C stage shown in Figs. 1 and 2 produce the following stage output data: I_{ds} (saturation current), $I_{ds\text{-}pk}$ (peak value of saturation current), R_{ds} (drain-source resistance), r_{62tgv} (slope of R_c characteristics), and r_{63tgv} (intercept of R_c characteristics). Inputs for this stage are initial measurements of I_{ds}, R_{ds}, R_c, R_{ch} (channel resistance), c_{62rfv} (slope of R_c characteristics), and c_{63rfv} (intercept of R_c characteristics).

4.2 Post-contact, Post-recess, to Post-gate (cr=>g or R)

The network for this stage model has eleven inputs plus the bias input. Six inputs are post-contact and five are post-recess. The network has 13 outputs, and 18 hidden neurons. The size of the hidden layer was again determined experimentally by varying the number of hidden neurons and selecting their number which resulted in the lowest training error over a number of training sessions. Graphs of the actual measured and modeled values for this stage are presented on Figs. 3 and 4. The graphs contain similar information as discussed earlier in Section 4.1 for the previous process stage.

Twelve of thirteen modeled outputs are depicted on the graphs. They are I_{ds} (drain to source saturation current), R_{ds} (drain-source resistance), R_{gs} (gate-source resistance), R_s (source resistance), R_d (drain resistance), V_{sat} (saturation voltage), $V_{br\text{-}dg}$ (breakdown voltage drain-to-

gate), V_{br-sg} (breakdown voltage source-to-gate), V_{po} (pinch-off voltage), G_m (mutual transconductance), I_{ds-pk} (peak value of the drain-to-source saturation current), e_a (electrical alignment). Inputs to this stage model are the following parameters: five of these are post-recess outputs listed earlier in Section 4.1, and six are post-contact inputs listed as inputs to the C stage models.

While the network-computed outputs on the training set are in very good to excellent agreement with the measurements, the test set response of MFFNN yields some differences between the measured and predicted outputs. It can be seen that I_{ds} values tend to be averaged over the set (see Fig. 4a). Prediction of R_s, R_{dg}, and R_d seem to be less than satisfactory. Prediction of the remaining parameters is rather accurate. Outliers among the test data, as expected, tend to produce larger errors.

5. Preliminary Conclusions from Current Modeling Efforts

A number of important issues emerged as the MFFNN training files were created. First of all, most of the substrate data or (s) stage as it is referred to in the work is obtained through destructive testing. That is, once characterization tests are performed, the wafer is of no further use in circuit fabrication. Therefore, only a few wafers per boule are characterized this way and their data is considered to be representative of the other wafers from the boule used for circuit fabrication. It was also discovered that these tests were not performed on every boule, therefore leaving some wafers without substrate characterization. This has been the case for the wafer which was selected for the preliminary modeling. The wafer for the preliminary modeling effort was selected at random. Care should be taken in the future work to ensure that the wafers used for network training have proper substrate characterization data.

The substrate measurements were taken on the bare substrate and are reported in millimeters. Since the measurements are taken prior to the actual fabrication, the data is not suited for coordinates XXYY,xxyy which is the structure identification scheme used for all other measurements. A routine needs to be created to convert this data in such a manner as to assign it to the MESFET which is closest to the point of measurement.

It has also been discovered that characteristics such as metalization thickness and width for the different process stages were measured at final stage. The test has been performed at the final stage, but these are characteristics which represent earlier stages and need to be shifted back to their respective stage models for earlier use. The reason the measurements were made at final rather than at the specific test stage is that testing takes a long time and slows down wafer fabrication. Therefore, since some characteristics are not affected by subsequent processing stages, their measurements are deferred until the final test.

The data manipulation routines are therefore currently being modified to allow the user to request specific characteristics rather than ones which have a certain time index of test. This is also being done so the user may request data on specific reticles within a wafer or data for the entire set of wafers. Once finished, complete data sets for modeling of each of the stages

for specific reticles or entire wafers will be readily available.

Similar modeling of the process stages discussed in this report will be performed for remaining stages. Once the data manipulation and statistical routines are completed, and the preliminary modeling results are analyzed, then most of the project focus will be shifted towards optimizing network architectures and minimizing the modeling errors for each processing stage.

6. Appendix: Paper in 1994 Int. IEEE Symposium on Circuits and Systems: **"Sensitivity Analysis for Minimization of Input Data Dimension for Feedforward Neural Network."**

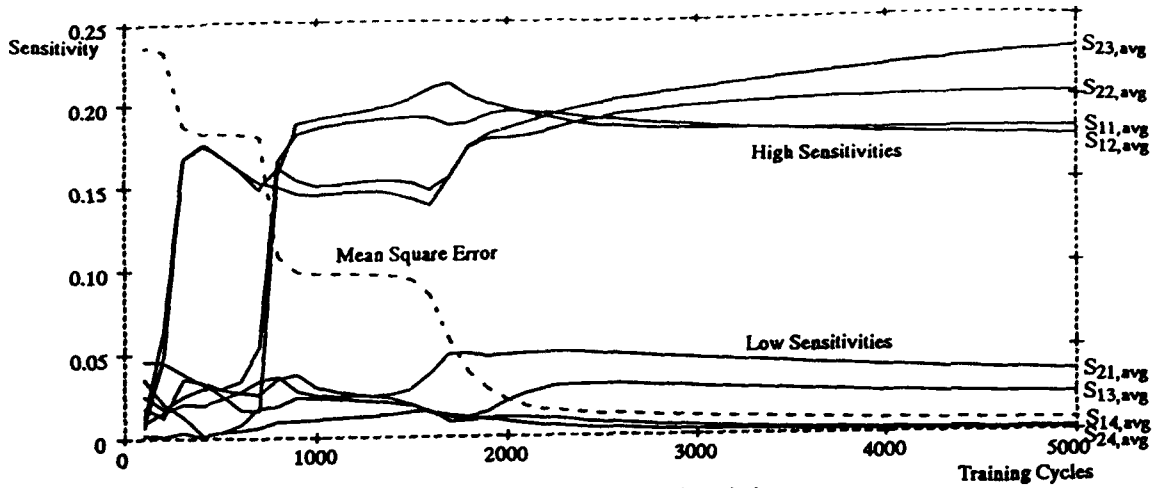


Fig. 1. Sensitivity profile during training for the full training set.

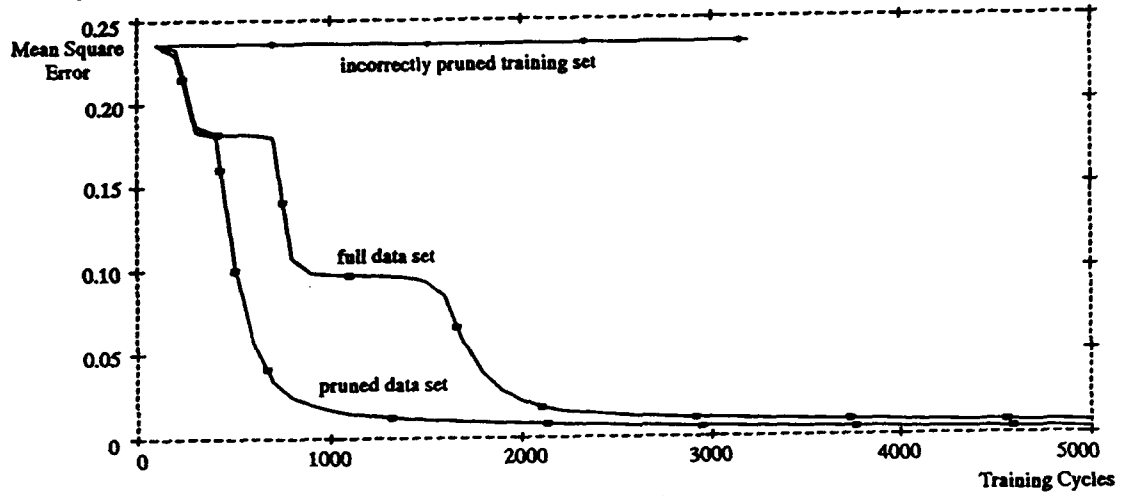


Fig. 2. Learning profile for full and pruned training sets.

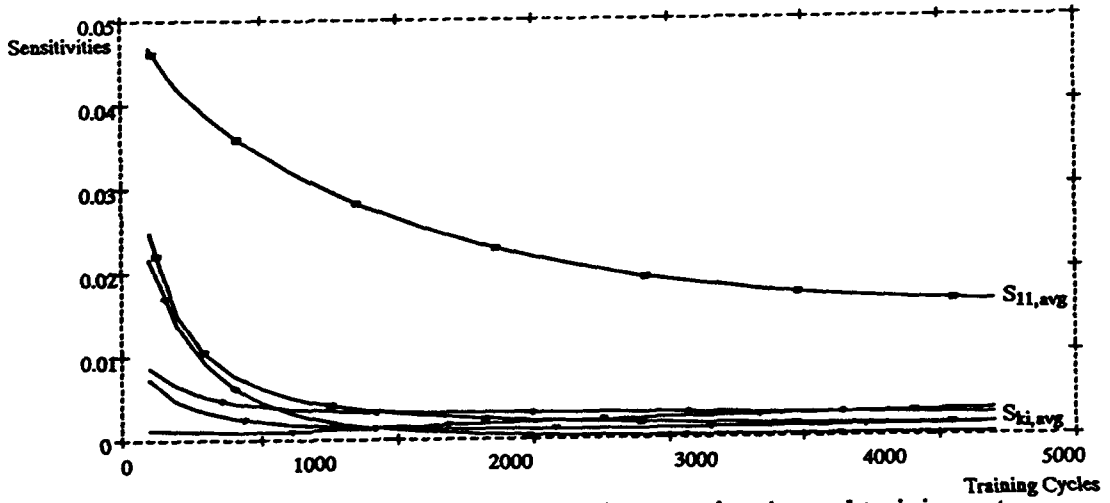


Fig. 3. Sensitivity profile during training for incorrectly trimmed training set.

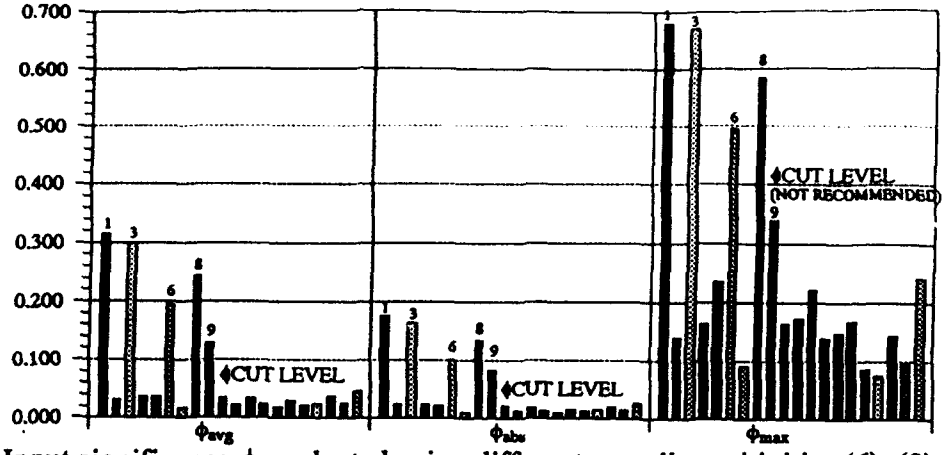


Fig. 4. Input significance ϕ evaluated using different overall sensitivities (6)–(8) and pruning criterion (16).

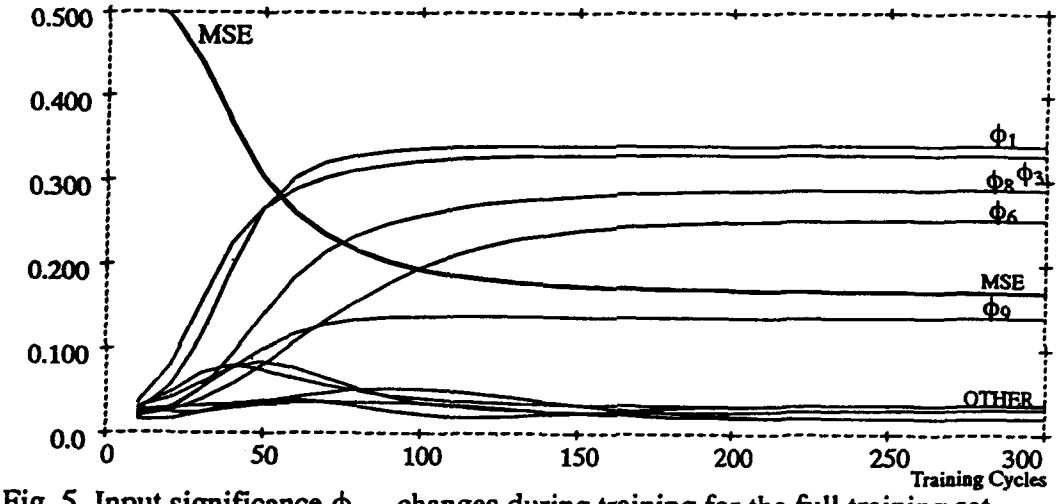


Fig. 5. Input significance ϕ_{avg} changes during training for the full training set.

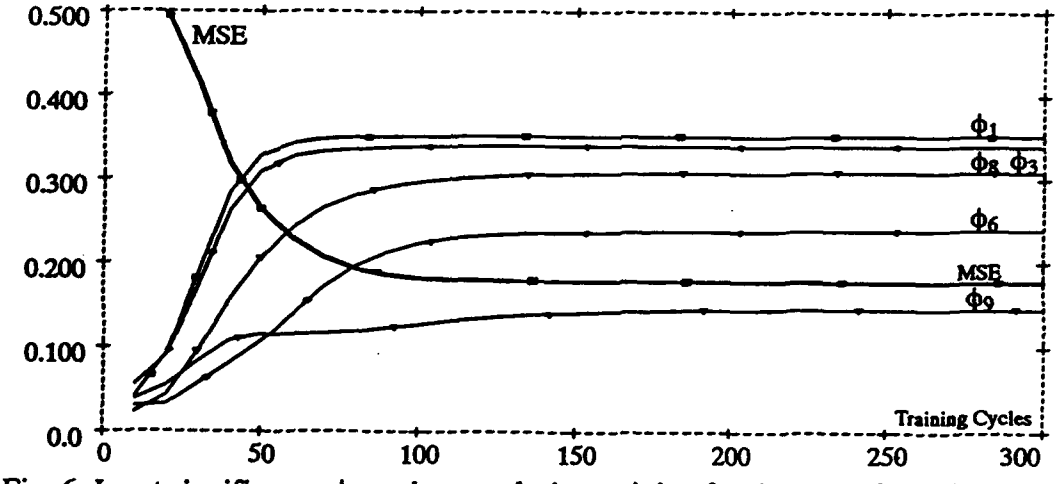
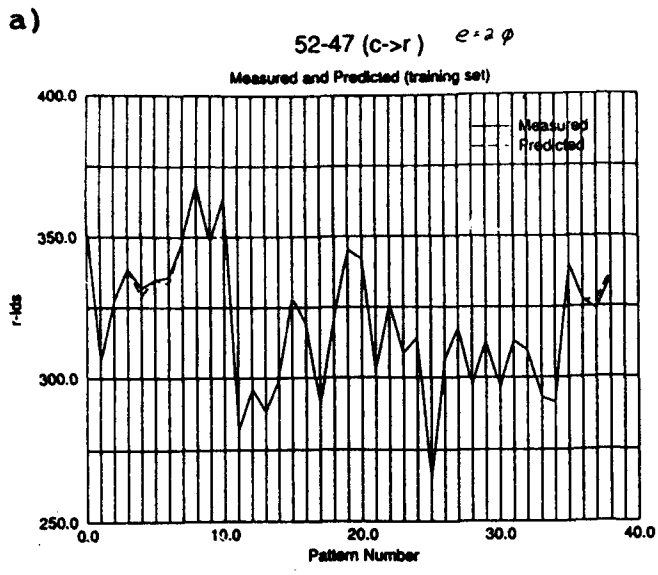
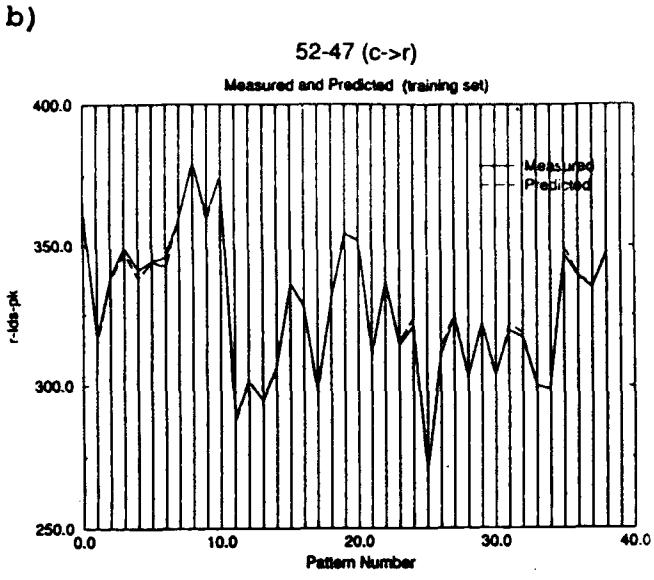
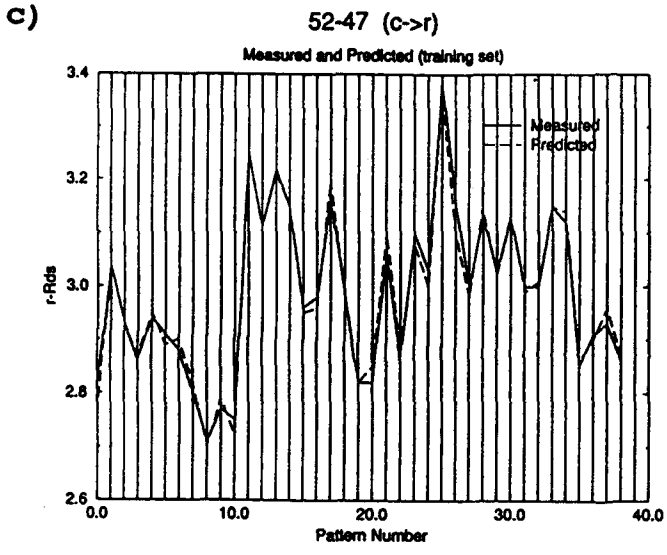
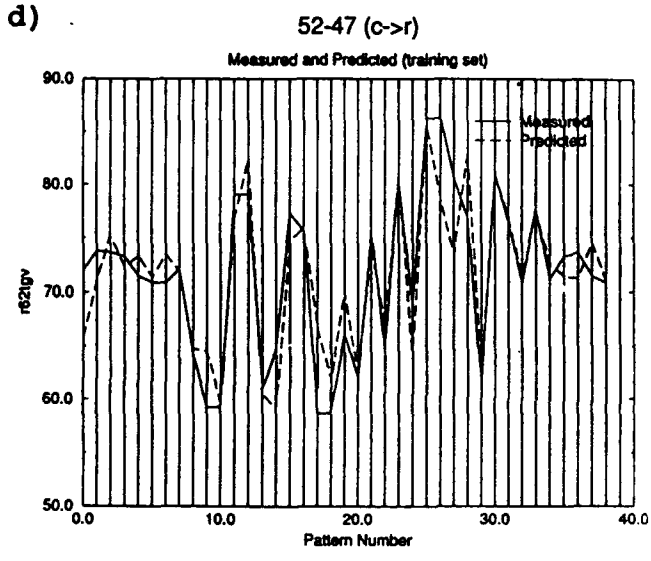


Fig. 6. Input significance ϕ_{avg} changes during training for the pruned training set.









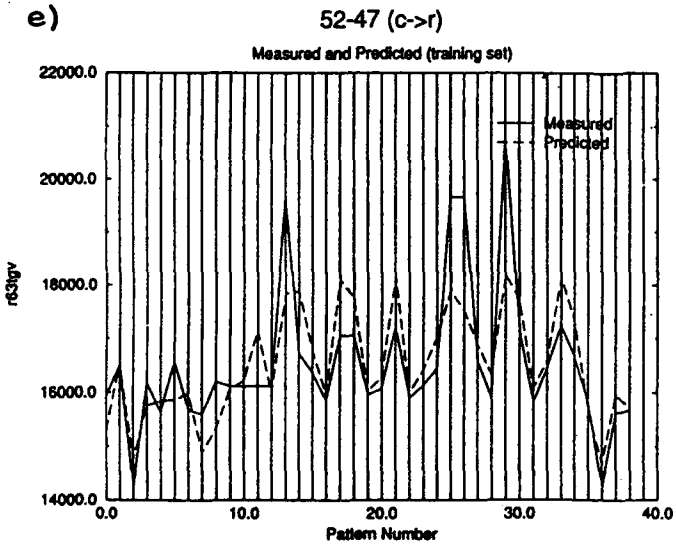


Fig. 1 Neural Network Model of the C Stage Tested on the Training Data

a) I_{ds}
b) $I_{ds\text{-}pk}$
c) R_{ds}
d) r_{62tgv}
e) r_{63tgv}

(see text for symbol explanations)

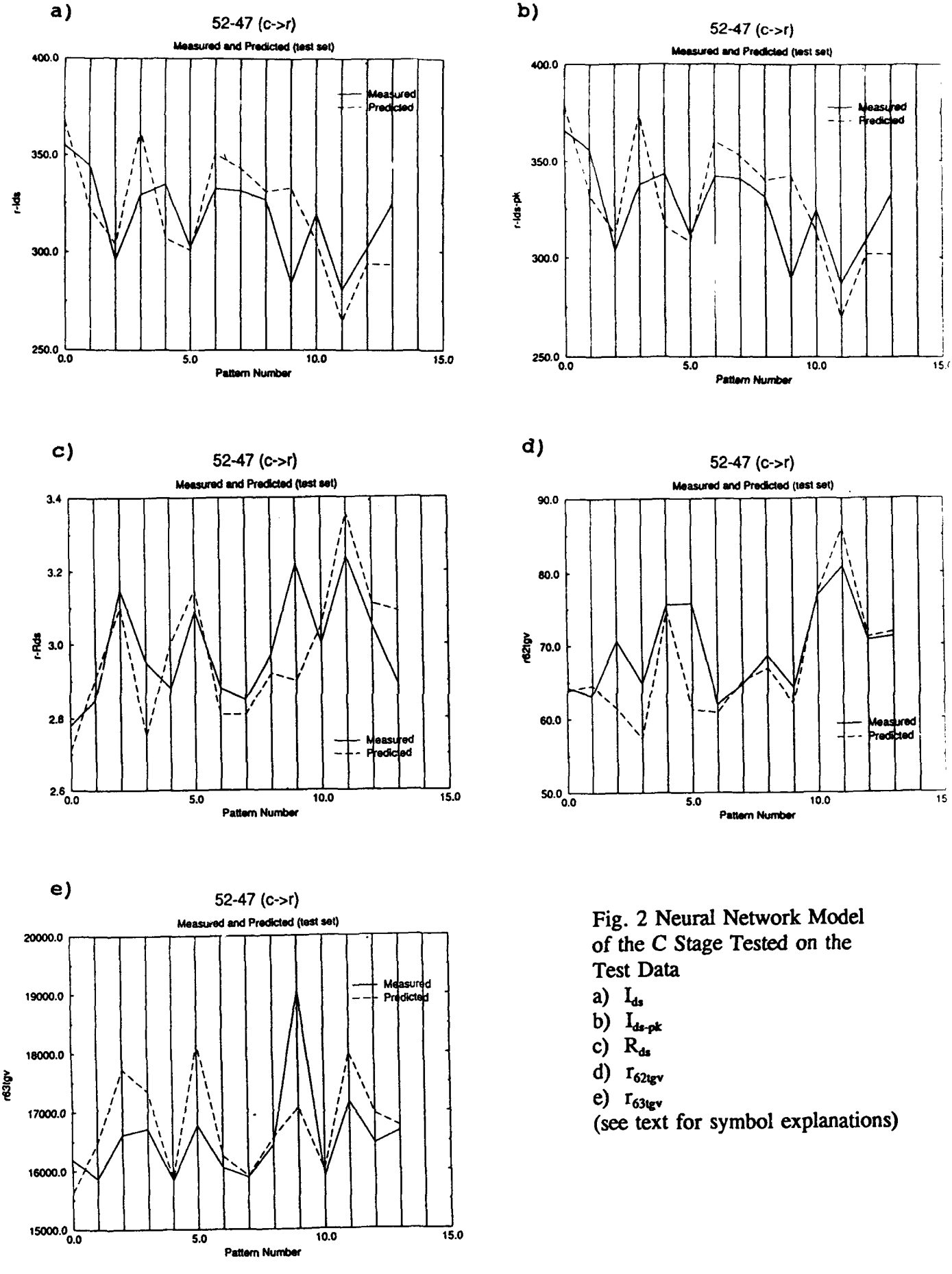


Fig. 2 Neural Network Model of the C Stage Tested on the Test Data

a) I_{ds}

b) $I_{ds\text{-}pk}$

c) R_{ds}

d) r_{62tgv}

e) r_{63tgv}

(see text for symbol explanations)

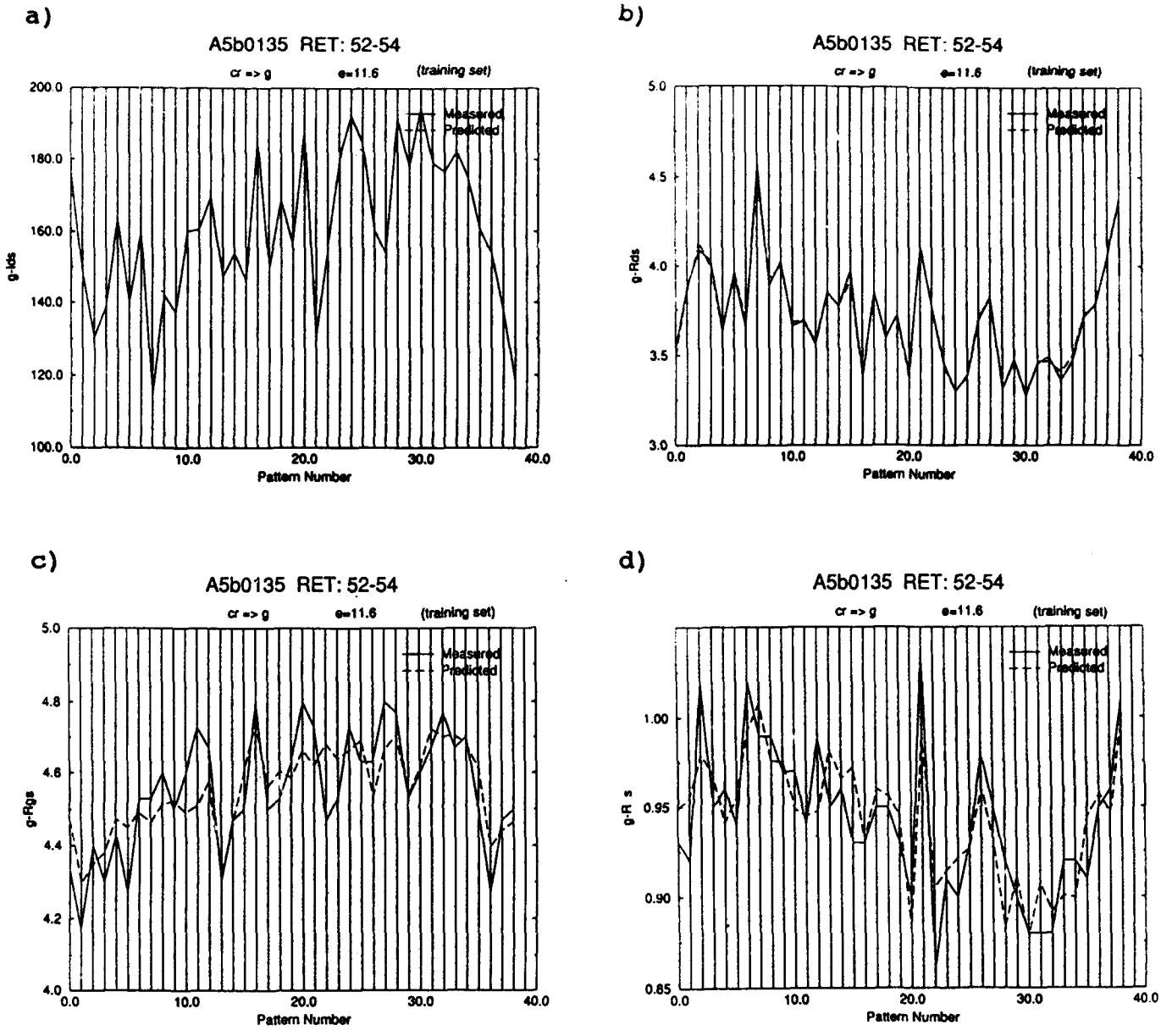


Fig. 3 Neural Network Model of the R Stage
Tested on the Training Data

a) I_{ds}
b) R_{ds}
c) R_{gs}
d) R_s

(see text for symbol explanations)

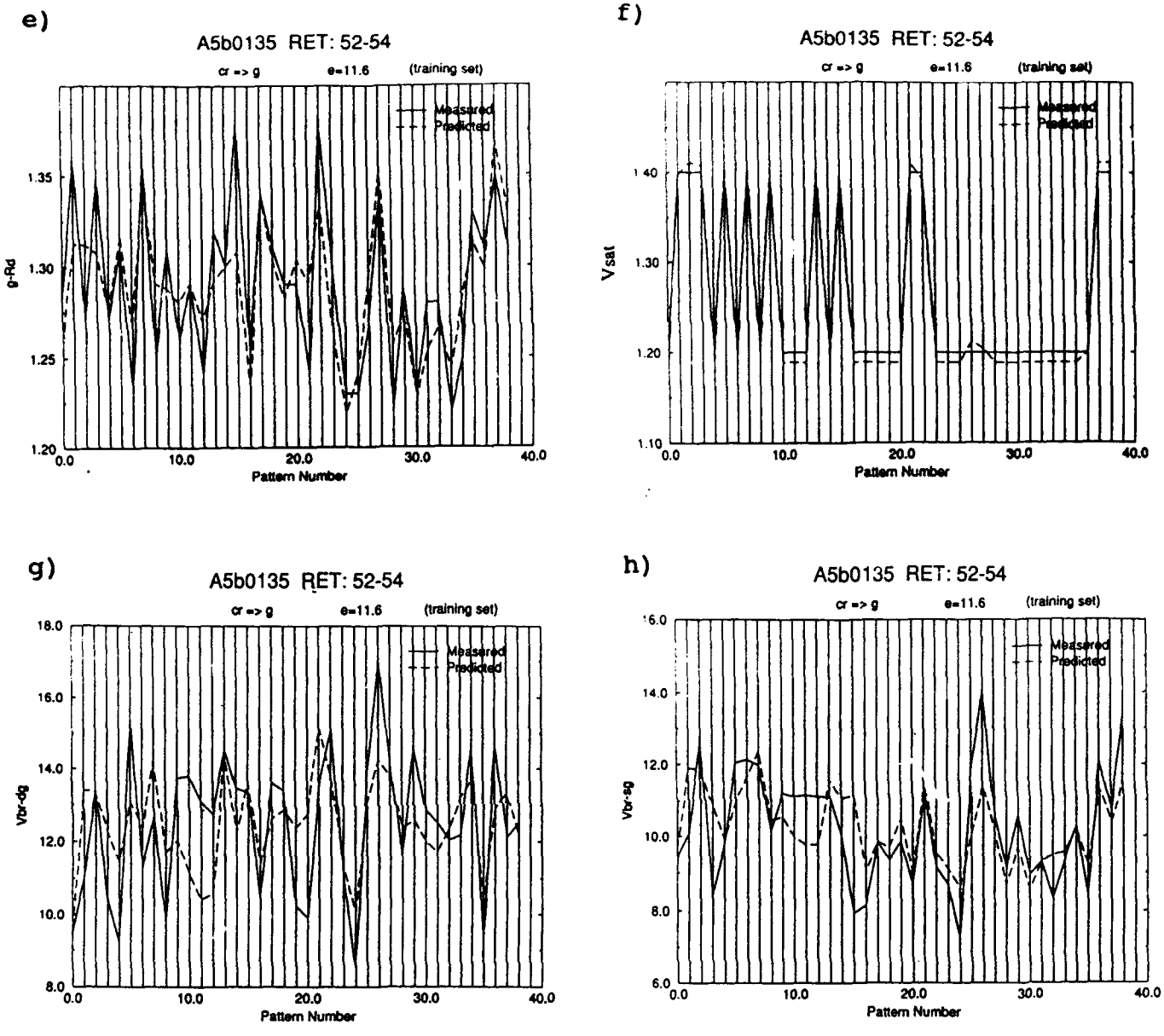


Fig. 3 Neural Network Model of the R Stage
Tested on the Training Data (ctd)

e) R_d
f) V_{sat}
g) $V_{br\text{-}dg}$
h) $V_{br\text{-}sg}$

(see text for symbol explanations)

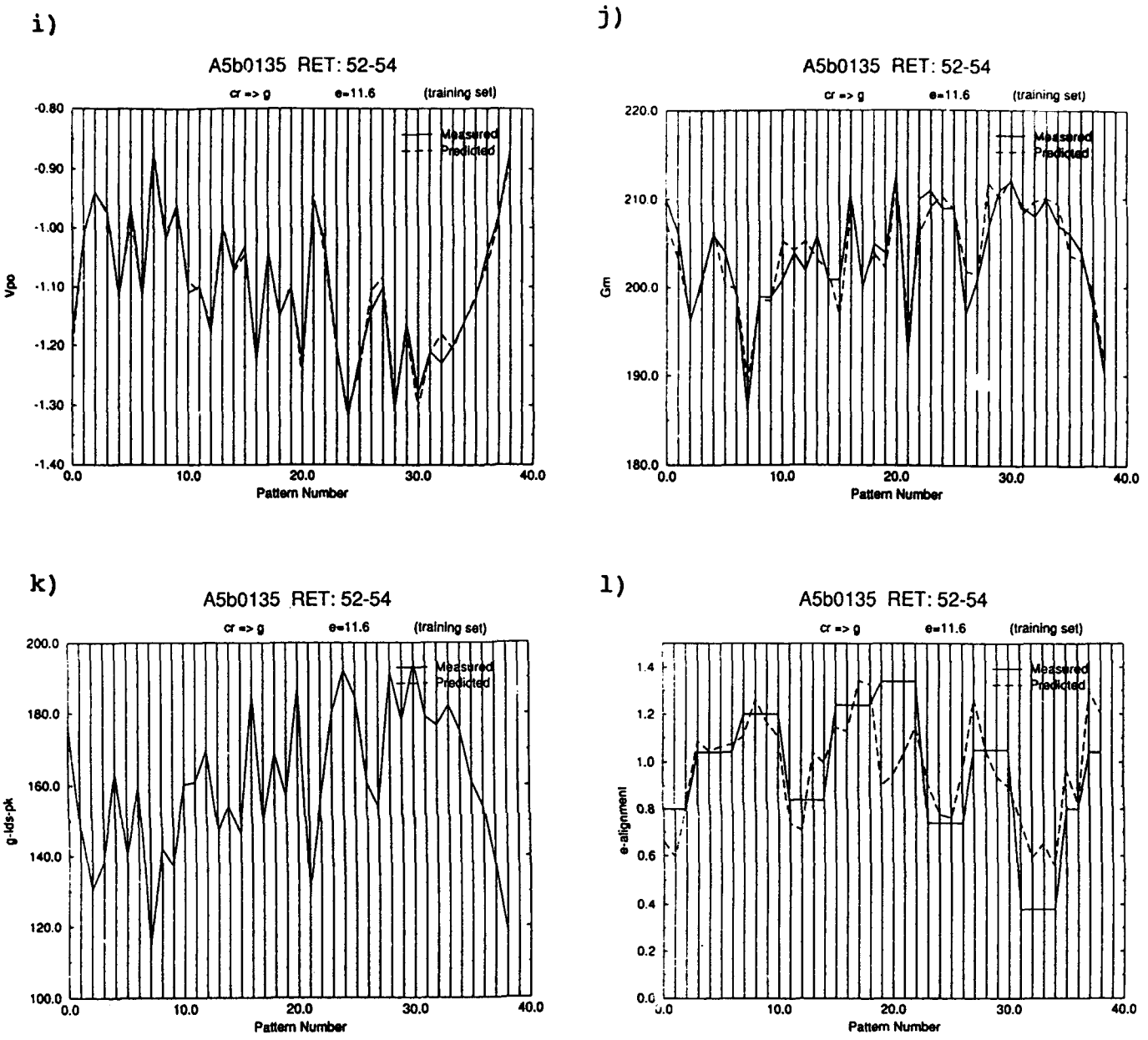


Fig. 3 Neural Network Model of the R Stage
Tested on the Training Data (ctd)

i) V_{po}
j) G_m
k) $I_{ds\text{-}pk}$
l) e_a

(see text for symbol explanations)

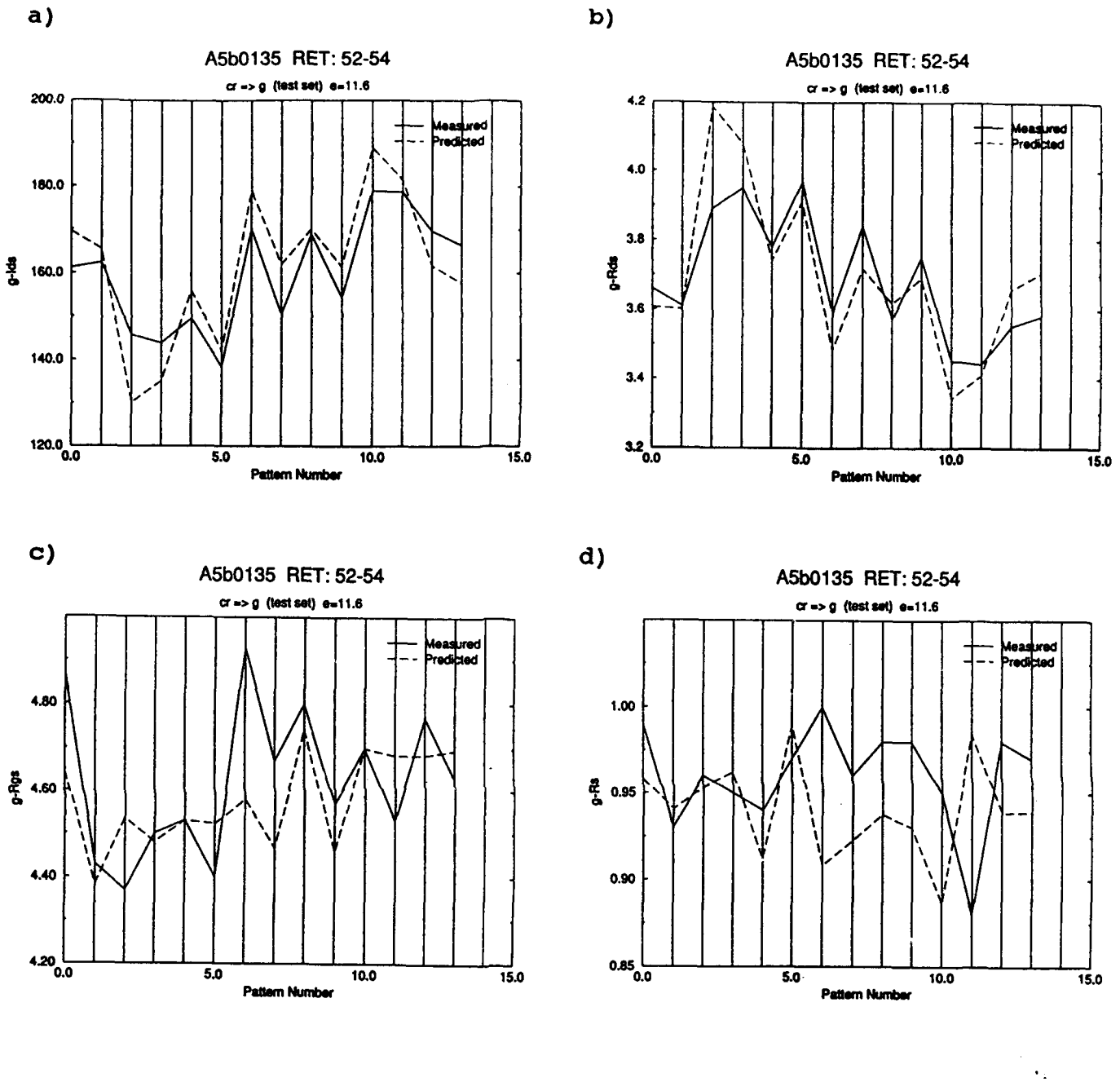
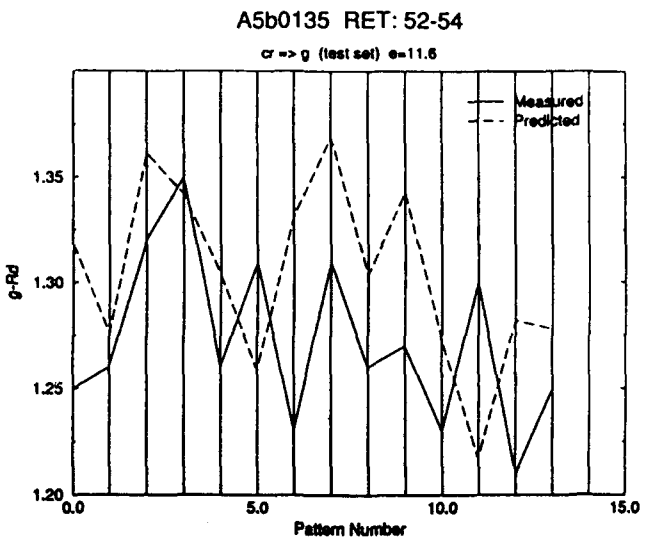


Fig. 4 Neural Network Model of the R Stage
Tested on the Test Data

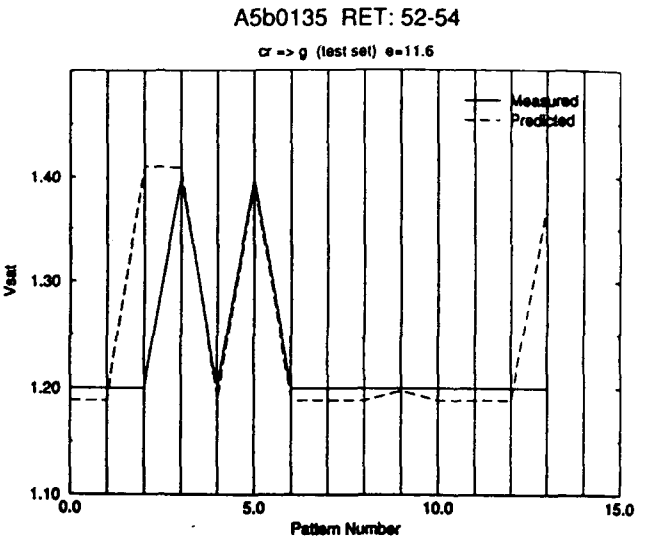
a) I_{ds}
b) R_{ds}
c) R_{gs}
d) R_s

(see text for symbol explanations)

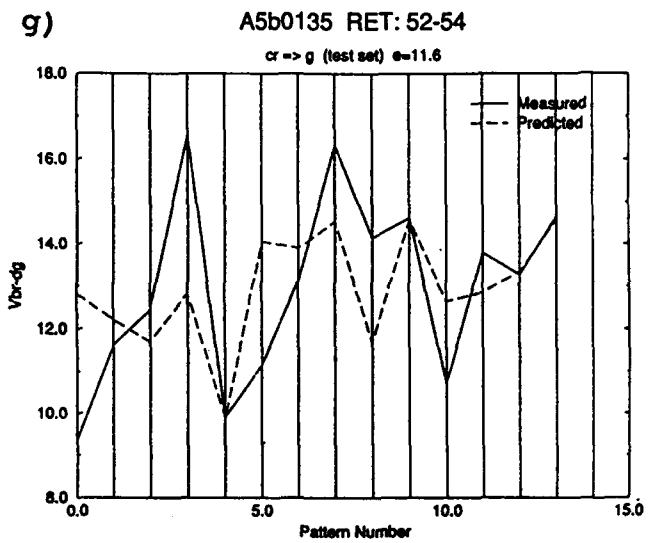
e)



f)



g)



h)

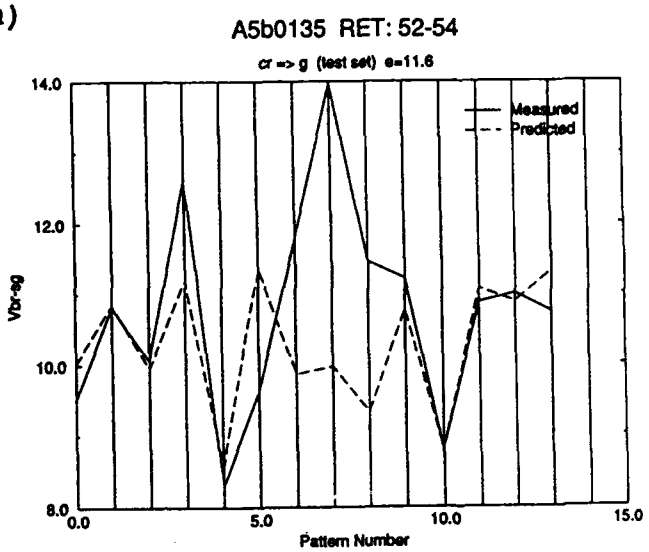


Fig. 4 Neural Network Model of the R Stage
Tested on the Test Data (ctd)

e) R_d
f) V_{sat}
g) $V_{br\text{-}dg}$
h) $V_{br\text{-}sg}$

(see text for symbol explanations)

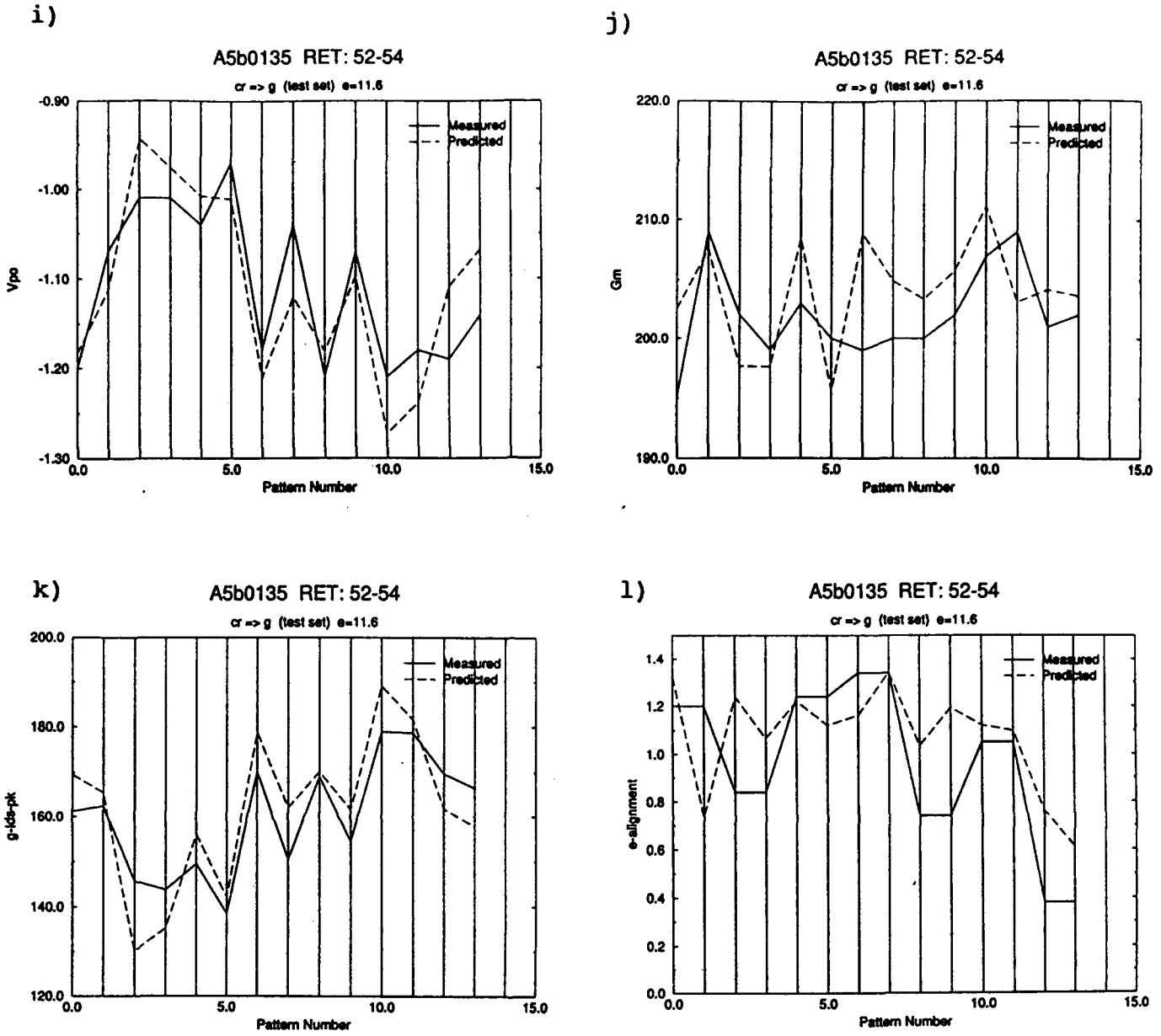


Fig. 4 Neural Network Model of the R Stage
Tested on the Test Data (ctd)

i) V_{po}
j) G_m
k) $I_{ds\text{-}pk}$
l) e_a

(see text for symbol explanations)

Sensitivity Analysis for Minimization of Input Data Dimension for Feedforward Neural Network

Jacek M. Zurada, Aleksander Malinowski, Ian Cloete♦
University of Louisville, Louisville Kentucky 40292, USA
e-mail: jmzura02@ulkyvx.louisville.edu
♦University of Stellenbosch, Stellenbosch, South Africa

Abstract: Multilayer feedforward networks are often used for modeling complex relationships between the data sets. Deleting unimportant data components in the training sets could lead to smaller networks and reduced-size data vectors. This can be achieved by analyzing the total disturbance of network outputs due to perturbed inputs. The search for redundant data components is performed for networks with continuous outputs and is based on the concept in sensitivity of linearized neural networks. The formalized criteria and algorithm for pruning data vectors are formulated and illustrated with examples.

Introduction

Neural networks are often used to model complex functional relationships between sets of experimental data. This is particularly useful when an analytical model of a process either does not exist or it is not known, but when sufficient data is available for embedding relationships existing between two or more data bases into a neural network model. Representative data can be used in such case to perform supervised training of a suitable neurocomputing architecture. Multilayer feedforward neural networks (MFNN) have been found especially efficient for this purpose [1, 2]. The minimization of redundancy in the training data is, however, an important issue and rather rarely addressed in the technical literature. MFNN considered here are trained using the popular error back-propagation technique in order to perform the feedforward process identification [3].

Let us consider a MFNN with a single hidden layer. The network performs a nonlinear and constrained mapping $\mathbf{o}=\Gamma(\mathbf{x})$, where $\mathbf{o}$ (Kx1), and $\mathbf{x}$ (Ix1) are output and input vectors, respectively. It is assumed that certain inputs bear none, or little, statistical or deterministic relationships to outputs and input vectors could therefore be compressed. The objective of this study is to reduce the dimensionality of the input vector, $\mathbf{x}$, and thus to prune the input data set, so that a smaller network can be utilized as a model of relationship between the data. Initial findings on this subject have been published in [4–6]. This paper introduces a more general and formal approach to reduction of input size of the network. The sensitivity approach can also be used to delete weights which are unimportant for neural network performance as it has been proposed in [7].

Sensitivities to Inputs

Let us define the sensitivity of a trained MFNN output, o_k, with respect to its input x_i as

$$S_{x_i}^{o_k} \doteq \frac{\partial o_k}{\partial x_i} \tag{1a}$$

which can be written succinctly as

$$S_{ki} \doteq S_{x_i}^{o_k} \tag{1b}$$

By using the standard notation of an error backpropagation approach [3], the derivative of (1a) can be readily expressed in terms of network weights as follows

$$\frac{\partial o_k}{\partial x_i} = o_k' \sum_{j=1}^{J-1} w_{kj} \frac{\partial y_j}{\partial x_i} \tag{2}$$

where y_j denotes the output of the j–th neuron of the hidden layer, and o_k' is the value of derivative of the activation function o=f(net) at the k–th output neuron. This further yields

$$\frac{\partial o_k}{\partial x_i} = o_k' \sum_{j=1}^{J-1} w_{kj} y_j' v_{ji} \tag{3}$$

where y_j' is the value of derivative of the activation function y=f(net) of the j–th hidden neuron ($y_J'=0$ since the J–th neuron is a dummy one, i.e. it serves as a bias input to the output layer). The sensitivity matrix **S** (KxI) consisting of entries as in (3) or (1b) can now be expressed using array notation as

$$\mathbf{S} = \mathbf{O}' \times \mathbf{W} \times \mathbf{Y}' \times \mathbf{V} \tag{4}$$

W (KxJ) and **V** (JxI) are output and hidden layer weight matrices, respectively, and **O'** (KxK) and **Y'** (JxJ) are diagonal matrices defined as follows

$$\begin{aligned} \mathbf{O}' &\doteq diag(o_1', o_2', \ldots, o_K') \\ \mathbf{Y}' &\doteq diag(y_1', y_2', \ldots, y_J') \end{aligned} \tag{5}$$

Matrix **S** contains entries S_{ki} which are ratios of absolute increments of output k due to the input i as defined in (1b). This matrix depends only upon the network weights as well as slopes of the activation functions of all neurons. Each training vector $\mathbf{x}^{(n)} \in \mathfrak{X}$, where $\mathfrak{X}=\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}\}$ denotes the training set, produces different sensitivity matrix $\mathbf{S}^{(n)}$ even for a fixed network. This is due to the fact that although weights of a trained network remain constant, the activation values of neurons

change across the set of training vectors $x^{(n)}$, n=1, 2, ..., N. This, in turn, produces different diagonal matrices of derivatives O' and Y', which strongly depend upon the neurons' operating points determined by their activation values.

Measures of Sensitivity over a Training Set

In order to possibly reduce the dimensionality of input vectors, the sensitivity matrix as in (4) needs to be evaluated over the entire training set 𝔛. Let us define the sensitivity matrix for the pattern x_n as $S^{(n)}$. There are several ways to define the overall sensitivity matrix, each relating to the different objective functions which need to be minimized.

The *mean square average sensitivities*, $S_{ki,\,avg}$, over the set 𝔛 can be computed as

$$S_{ki,\,avg} \doteq \sqrt{\frac{\sum_{n=1}^{N}\left(S_{ki}^{(n)}\right)^2}{N}} \tag{6}$$

Matrix S_{avg} (KxI) is defined as $[S_{avg}]=S_{ki,avg}$. This method of sensitivity averaging is coherent with the goal of network training which minimizes the mean square error over all outputs and all patterns in the set.

The *absolute value average sensitivities*, $S_{ki,\,abs}$, over the set 𝔛 can be computed as

$$S_{ki,\,abs} \doteq \sqrt{\frac{\sum_{n=1}^{N}\left|S_{ki}^{(n)}\right|}{N}} \tag{7}$$

Matrix S_{abs} (KxI) is defined as $[S_{abs}]=S_{ki,abs}$. Note that summing sensitivities across the training set requires taking their absolute values due to the possibility of cancelations of negative and positive values. This method of averaging may be better than (6) if sensitivities $S_{ki}^{(n)}$, n=1, ..., N, are of disparate values.

The *maximum sensitivities*, $S_{ki,\,max}$, over the set 𝔛 can be computed as

$$S_{ki,\,max} \doteq \max_{n=1..N}\left\{S_{ki}^{(n)}\right\} \tag{8}$$

Matrix S_{max} (KxI) is defined as $[S_{max}]=S_{ki,max}$. This sensitivity definition allows to prevent pruning inputs which are relevant for the network only in small percentage of input vectors among the whole training set. However, it can happen that a few fuzzy patterns in a large set can affect entires of sensi-

sensitivity array by associating fuzziness with additional inputs. Those fuzzy results are masked in such case by averaging in (6)–(7), and not by (8). Therefore the significance of inputs can be overestimated and therefore some unimportant inputs may remain after reducing the dimension.

Any of the sensitivity measure matrices proposed in (6)–(8) can provide useful information as to the relative significance of each of the inputs in $\mathfrak{X}$ to each of the outputs. For the sake of simplicity, however, only matrix defined in (6) will be used in further discussion. The cumulative statistical information resulting from (6) will be used along with criteria for reducing the number of inputs to the smallest number sufficient for accurate learning. These criteria are formulated in the next section.

Criteria for Pruning Inputs

Inspection of the average sensitivity matrix S_{avg} allows to determine which inputs affect outputs least. A small value of $S_{ki,avg}$ in comparison to others means that for the particular k–th output of the network, the i–th input does not significantly contribute to output k, and may therefore be possibly disregarded. This reasoning and results of experiments allow to formulate the following practical rule: *The sensitivity matrices for a trained neural network can be evaluated for both training and testing data sets; the values of average sensitivity matrix entries can be used for determining the least significant inputs and for reducing the size of network accordingly through pruning unnecessary inputs.*

When that one or more of the inputs have relatively small sensitivity in comparison to others, the dimension of neural network can be reduced by dropping them, and smaller–size neural network can be successfully retrained in most cases. The criterion used in this paper for determining which inputs can be pruned is based on so called the largest gap method.

In order to normalize the data relevant for comparison of significance of inputs, the sensitivity matrix defined in (6)–(8) has to be additionally preprocessed. The formulas often used for scaling are given in (9) and map each input into range [0 ; 1] and each output output into range [–1 ; 1]:

$$\hat{x}_i^{(n)} \doteq \frac{x_i^{(n)} - \min\limits_{n=1\_N}\left\{x_i^{(n)}\right\}}{\left(\max\limits_{n=1\_N}\left\{x_i^{(n)}\right\} - \min\limits_{n=1\_N}\left\{x_i^{(n)}\right\}\right)}, \qquad \hat{o}_k^{(n)} \doteq \frac{o_k^{(n)} - \frac{\left(\max\limits_{n=1\_N}\left\{o_k^{(n)}\right\} + \min\limits_{n=1\_N}\left\{o_k^{(n)}\right\}\right)}{2}}{\left(\max\limits_{n=1\_N}\left\{o_k^{(n)}\right\} - \min\limits_{n=1\_N}\left\{o_k^{(n)}\right\}\right)} \tag{9}$$

If input and output data scaling (9) was performed before network training, no additional operations on S_{ki} is required and we have

$$\hat{S}_{ki,avg} \doteq S_{ki} \tag{10}$$

Note that the scaling can be performed either on entries of S or S_{avg}. Experiments were performed also for scaling inputs into range [−1 ; 1]. Similar results were achieved for the same learning conditions. The latter scaling seems to fasten the learning convergence while accuracy and relations among sensitivities remain unchanged.
In case when network original inputs and outputs are not scaled to the same level, additional scaling (11) is necessary to allow for accurate comparison among inputs.

$$\hat{S}_{ki,avg} \doteq S_{ki} \frac{\left(\max_{n=1\_N} \left\{ x_i^{(n)} \right\} - \min_{n=1\_N} \left\{ x_i^{(n)} \right\} \right)}{\left(\max_{n=1\_N} \left\{ o_k^{(n)} \right\} - \min_{n=1\_N} \left\{ o_k^{(n)} \right\} \right)} \tag{11}$$

The significance of i–th input Φ_i across the entire set $\mathfrak{X}$ is defined as:

$$\Phi_{i,avg} \doteq \max_{k=1\_K} \left\{ \hat{S}_{ki,avg} \right\} \tag{12}$$

Φ_{abs} and Φ_{max} can be evaluated similarly to Φ_{abs} defined in (12). In order to distinguish inputs with high and low importance, entries of Φ_i have to be sorted in descending order so that:

$$\Phi_{i_{m+1}} \geq \Phi_{i_m}, \quad m = 1, \ ..., \ I - 1 \tag{13}$$

where i_m is a sequence of sorted input numbers. Let us define the measure of gap as (14)

$$g_{i_m} \doteq \frac{\Phi_{i_m}}{\Phi_{i_{m+1}}} \tag{14}$$

and then find the largest gap using the formula (15).

$$g_{MAX} \doteq \max_{i_m} \left\{ g_{i_m} \right\} \quad \text{and} \quad m_{CUT} \doteq m \text{ such that } g_{i_m} = g_{MAX} \tag{15}$$

If condition (16) is valid, then the found gap between m_{CUT} and m_{CUT+1} is large enough.

$$C g_{MAX} > \max_{i_m \neq i_{m_{CUT}}} \left\{ g_{i_m} \right\} \tag{16}$$

Constant C from (16) is chosen arbitrary within a reasonable range (e.g. C=0.5. The smaller C the stronger is condition for existence of the acceptable gap.) All inputs with index $\{i_{m+1}..i_{I-1}\}$ can be pruned with the smallest loss of information to the MFNN.

The gap method can be also applied for comparison among sensitivities of inputs to each output separately. For this purpose, a set containing candidates for pruning can be created for every output. Final pruning is performed by removing these inputs which can be found in every set determined previously for each output independently.

Certainly, S_{avg} can be evaluated meaningfully only for well trained neural networks. Despite this disadvantage, proposed criteria can still save computational effort when initial learning can be performed on smaller, but still representative subset of data. S_{avg} can be evaluated based either on data set used for initial training or on complete data set. Subsequently, newly developed neural network with appropriate inputs can be retrained using the full set of training patterns with reduced dimension.

Numerical Examples

A series of numerical simulations was performed in order to verify the proposed definitions and the pruning criteria. In the first experiment a training set for a neural network was generated using four inputs $x_1..x_4$ and two outputs o_1 and o_2. Values of outputs were correlated with x_1 and x_2 for o_1, and with x_2 and x_3 for o_2. Input vectors $\mathbf{x}$ (4x1) were produced using a random number generator. The expected values of vector $\mathbf{d}$ (2x1) for the output vector $\mathbf{o}$ (2x1) were evaluated for each $\mathbf{x}$ using a known relationship $\mathbf{d}=\mathbf{F}(\mathbf{x})$ where $\mathbf{d}$ is the desired (target) output vector for supervised training. The training set $\mathfrak{X}$ consisted of N=81 patterns. A neural network with 4 inputs, 2 outputs and 6 hidden neurons (I=5, J=7, K=2) has been trained for the mean square error defined as in (17)

$$MSE \doteq \sqrt{\frac{\sum_{n=1}^{N}\sum_{k=1}^{K}\left(d_k^{(n)} - o_k^{(n)}\right)^2}{N}} \tag{17}$$

equal 0.001 per input vector. Matrices of sensitivities were subsequently evaluated and S_{avg} produced at the end of training over the entire input data set $\mathfrak{X}$.

The changes of sensitivity entries during learning are presented in Fig. 1. It can be seen that an untrained neural network in the example has per average smaller sensitivities than after the training. During the training some of the average sensitivities $S_{ki,avg}$ increase, while other converge towards low values. Final values of sensitivities of the first output offer hints for deleting x_3 and x_4, and these for the second output indicate that x_1 and x_4 could be deleted. The only input which shows up in both sets candidates for deletion is x_4. Therefore, the fourth input to the network can be skipped and its dimension reduced to 3 (I=4).

After deleting x_4 from the learning data set the new network with 3 inputs was trained successfully with the same accuracy. The learning profiles for full and reduced input sets for the same

learning conditions are compared in Fig. 2. Not only the network with 3 inputs trains within smaller number of cycles, but each learning cycle is performed quicker due to the reduced input layer size.

If an input not recommended for pruning is erroneously deleted, the network was found unable to learn the data sets. The mean square error per pattern has remained at the level of approximately 0.25 as it is shown in Fig. 2. The entries of the sensitivity matrix remain at low level as it is shown in Fig. 3. There may still be some gap between entries, but it cannot be used for pruning because the MFNN has not learned vectors correctly and after input dimension reduction would not be able to learn more accurate. Gap which can be seen in the Fig. 3 has a meaning that for the insufficient accuracy which was achieved during the training, only one input could be left in the network without significant deterioration of performance.

The second experiment was performed using larger network and fuzzy data. MFNN had 20 inputs (I=21), 10 hidden neurons (J=26) and 4 outputs (K=4). There were N=500 patterns in the training set and several additional data sets of the same size for network performance evaluation. The network was successfully trained to the MSE error of 0.15. However, due to the fuzziness of the data MSE error for additional sets remained at the level of 0.20.

All outputs were strongly correlated with inputs x_1, x_2, x_3, x_4, x_6, x_8, and x_9. Input x_6 during data generation was multiplied by random numbers, while the influence of x_2 and x_4 on outputs was scaled down to remain small in comparison to other inputs (less than 0.05).

The input importances calculated using formulas (6)–(8) are shown in Fig. 4. Inputs x_2 and x_4 are placed even after sorting as less important than some of them which are not correlated at all. This occurred because of their low correlation to outputs, and they can be ignored as well as other not correlated for given MSE error as a final condition for training. The sequence of significance is the same for all proposed methods, however, the size of gaps are different in each case. Value C=0.5 prevents pruning using ϕ_{max} definition. Note that the maximum method does not give the clear clue where to set the level for purging due to fuzziness of the training data.

The result of initial training is shown in Fig. 5. It can be determined from this figure which inputs should remain after pruning. The network performance after pruning is shown in Fig. 6. No additional dimension reduction is possible because no large gap in input importances can be found. The speed of training has increased mostly because of reduction of the MFNN size (input dimension reduced by 4). The necessary number of cycles for training has also decreased, but not so dramatically as in the first experiment.

Conclusions

Using the sensitivity approach for input layer pruning seems particularly useful when network training requires large amount of redundant data. In the first phase, network can be pre-trained

until the training error decreases satisfactorily. Then sensitivity matrices can be evaluated and dimension of the input layer possibly reduced. Learning can subsequently be resumed until the training error reduces to acceptable low value. This process can be repeated, however, usually only the first execution yields significant improvement. Numerical experiments indicate that the effort of additional network retraining can be too high in comparison to benefits of further minimization.

Should the redundancy in training data vectors exist, the proposed approach based on the average sensitivity matrices for input data pruning allows for more efficient training. This can be achieved at a relatively low computational cost and based on heuristic data pruning criteria outlined in the paper. The approach can be combined with other improved training strategies such as increased complexity training [5]. Extension of the proposed sensitivity–based input pruning concept beyond continuous output values seems desirable for case of networks with binary outputs such as classifiers and other binary encoders.

Bibliography

[1] K. M. Hornik, M. Stinchcombe, H. White, "Multilayer Feedforward Networks Are Universal Approximators", Neural Networks, vol. 2, 1989, pp.359–366.

[2] K. I. Funanashi, "On the Approximate Realization of Continuous Mappings by Neural Networks", Neural Networks, vol. 2, 1989, pp. 183–192.

[3] J. M. Zurada, "Introduction to Artificial Neural Systems", West Publishing Company, St. Paul, Minn., 1992.

[4] L. Fu, T. Chen, "Sensitivity Analysis for Input Vector in Multilayer Feedforward Neural Networks", Proc. of IEEE International Conference on Neural Networks, San Francisco, CA, March 28–April 1, 1993, vol. 1, pp.215–218.

[5] I. Cloete, J. Ludik, "Increased Complexity Training", Proc. of IWANN'93, Sitges, Spain, June 9–11, 1993.

[6] J. M. Zurada, A. Malinowski, I. Cloete, "Sensitivity Analysis for Pruning of Training Data in Feedforward Neural Networks", to be published in Proc. of First Australian and New Zeland Conference on Intelligent Information Systems, Perth, Western Australia, December 1–3, 1993.

[7] E. D. Karnin, "A Simple Procedure for Pruning Back–Propagation Trained Neural Networks", IEEE Trans. on Neural Networks, vol. 1, No. 2, June 1990.